PUREGYM

FINAL REPORT

Team 11

Institution: London School of Economics

Course: Data Analytics Career Accelerator

Due Date: 27th August, 2024

1. Background of the Business

Introduction

PureGym Limited is a high-value gym chain representing the highest British network by membership, with over 2 million members. To enhance its offerings, the company aims to deliver an affordable, high-quality product. However, it is crucial to determine if their current product and class offerings provide the best value to customers. Specifically, the focus is on the suitability and optimization of off-peak products and the effectiveness of class offerings across its various locations.

Business Problem

Concerns have been raised about the need for PureGym to improve its product and class offerings to meet member expectations and enhance their experience.

To better understand the situation, an analysis of member behaviour and class popularity across selected gyms is essential. This report will provide insights from internal data on the work out areas, member visits, class attendance and its timetables aiming to address key questions about;

- Which classes had the highest and lowest participation status; according to gender, age, time of the day?
- Which days experience the most significant and least attendance? Is there any comparison between weekdays and weekends?
- Which classes are underperforming and require to be reduced or replaced?
- How does the time of day impact class capacity utilisation?
- How does the distribution of check-in times vary across different days?
- Are there any opportunities for improvement in lowest utilised classes by percentage filled and total attendance

The approach focuses on optimising attendance by reducing cancellations and waitlists, which in turn helps minimise no-shows.

2. Project Development Process

This section outlines the overall approach and methodologies used in the project. It emphasises the following:

Data Cleaning and Preparation

The project began with extracting and inspecting the provided datasets, followed by a thorough data cleaning process to ensure the reliability of the subsequent analysis. We addressed missing values by imputing the mean, particularly in attendance and waitlist columns, to maintain the dataset's integrity without introducing bias. Outliers were identified using Z-scores and the Interquartile Range (IQR) and were removed to prevent skewing the results. Normalisation was carried out to ensure comparability across variables like class sizes and waitlists, enabling balanced analysis.

The datasets were then categorised to facilitate targeted analysis, grouping classes by popularity, cancellation rates, and no-show frequencies. This segmentation allowed us to identify specific patterns and trends, providing a clearer understanding of the data.

Data Analysis Approach

Python was chosen for its robust data analysis and visualisation capabilities, utilising libraries such as Pandas for data manipulation. We began by segmenting members according to demographics and examining their engagement with various products and classes.

Descriptive statistics were employed to summarise key metrics such as attendance, cancellations, and no-shows, establishing a foundational understanding of the data. Comparative analysis was then conducted to uncover trends across different age groups, guiding our recommendations for product optimisation and class schedule adjustments. For instance, by analysing class preferences by age group, we were able to tailor recommendations to better align with member interests, thereby improving attendance.

Correlation analysis explored relationships between key variables. This analysis revealed significant insights, such as the correlation between high waitlists and no-show rates, indicating that overbooking was likely an issue in popular classes. Understanding this was crucial for identifying inefficiencies in class utilisation.

To predict the impact of potential scheduling changes on attendance, we applied linear regression models. These models quantified the effects of various adjustments, providing a data-driven foundation for our recommendations. For example, identifying peak demand hours allowed us to suggest rescheduling classes to times that would likely increase utilisation rates.

Visualisation Decisions

Visualisation was key to effectively communicating our findings. We used Python's Matplotlib library to create a variety of graphs, each tailored to represent specific data insights clearly:

1. Stacked Bar Chart: This visualisation illustrated class capacity utilisation on Saturdays, showing both average attendance and remaining capacity. Annotating bars with utilisation percentages provided clarity, making it easier to assess the effectiveness of class scheduling.
2. Histogram for Check-In Times: A histogram visualised the distribution of check-in times on Wednesdays, using hourly bins to highlight peak check-in hours. This provided valuable insights for optimising resource allocation and staffing.
3. Dual Axis Bar and Line Plot: This combined plot displayed the percentage of class capacity filled alongside the total number of participants, offering a comprehensive view of class performance and highlighting discrepancies between high attendance and low utilisation.
4. Simple Bar Chart for Participation Status: Simple bar charts illustrate the proportions of no-shows, cancellations, and waitlist occurrences for the top classes. These visualisations provided clear insights into customer behaviour, identifying areas for improvement such as overbooking or managing high-demand classes.

Colour choices were made with accessibility in mind, using intuitive and clear schemes like blue for no-shows and yellow for waitlists. The visualisations were designed for clarity and readability, ensuring a logical flow from problem identification to solutions. Accessibility was also prioritised, with clear labelling and consistent design elements, making the data understandable to all users.

This structured approach to data extraction, cleaning, analysis, and visualisation enabled us to deliver actionable insights and recommendations aligned with PureGym's operational goals.

3. Technical Overview of the Code

This section is more specific and focuses on the implementation details of the project using Python. It includes:

Tools and Libraries

Python was chosen for its versatility and powerful libraries, making it ideal for data transformation and detailed analysis. Key libraries used included Pandas for data manipulation and Matplotlib for visualisation.

Data Manipulation with Pandas

Pandas was used to perform essential tasks like grouping, aggregating, and merging datasets. These operations were crucial for calculating key metrics such as total attendance, cancellations, and no-shows. Functions like groupby and aggregation allowed efficient computation of these statistics, providing insights into customer behaviour and class popularity.

Advanced Analytical Techniques

Advanced techniques, such as calculating proportions of different participation statuses and class capacity utilisation, were used to gain deeper insights into customer engagement. Python's flexibility allowed for seamless execution of these complex calculations, facilitating the exploration of various scenarios and hypotheses.

Merging Datasets

Integrating multiple datasets was critical for providing a comprehensive view of participation metrics. By merging data such as class schedules, attendance records, and other relevant information, the analysis could correlate attendance patterns with specific class attributes, optimising class offerings and schedules.

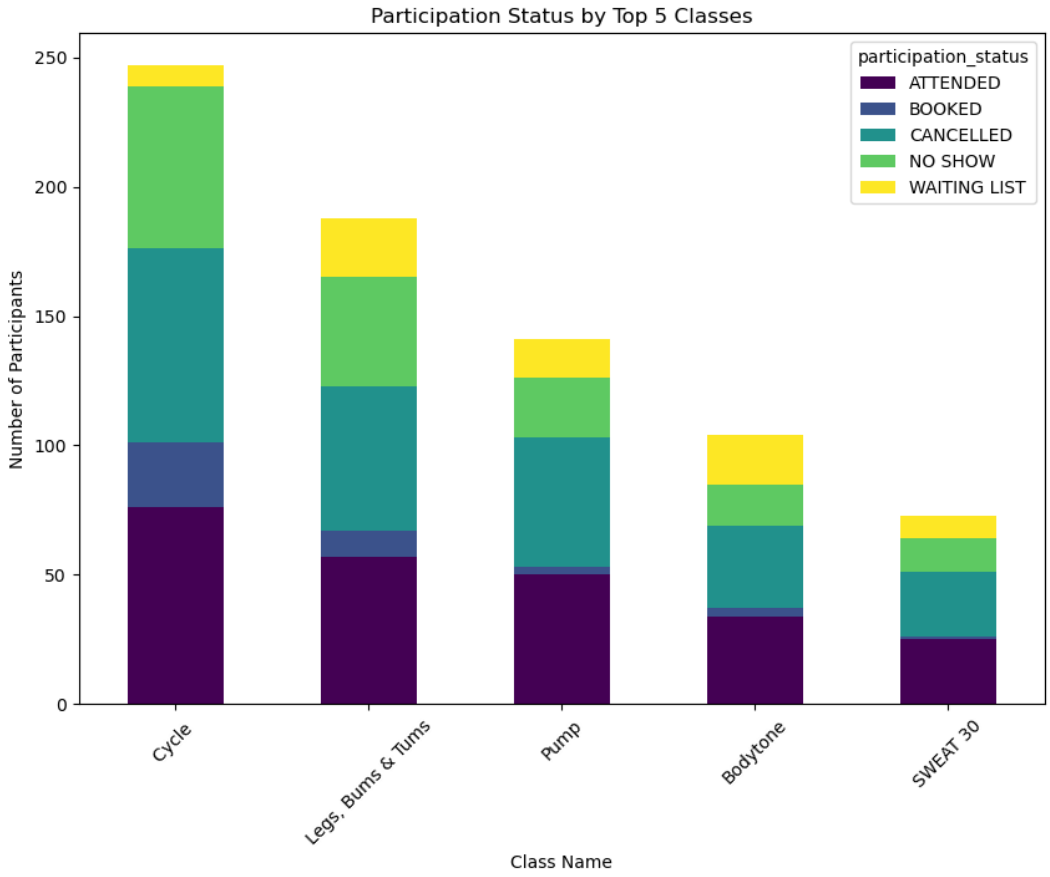
Visualisation Techniques with Matplotlib

Matplotlib was used to create a variety of graphs, each tailored to effectively represent specific data insights. Stacked bar charts, histograms, dual-axis plots, and simple bar charts were used to illustrate class capacity utilisation, check-in time distributions, and participation status proportions. These visualisations provided clear insights and facilitated a nuanced understanding of the data.

By focusing on the practical tools and specific coding techniques, the Technical Overview of the Code provides a detailed look at how data analysis and visualisation were implemented. It outlines the technical choices and processes behind the analysis, contributing to the project's success in delivering actionable insights.

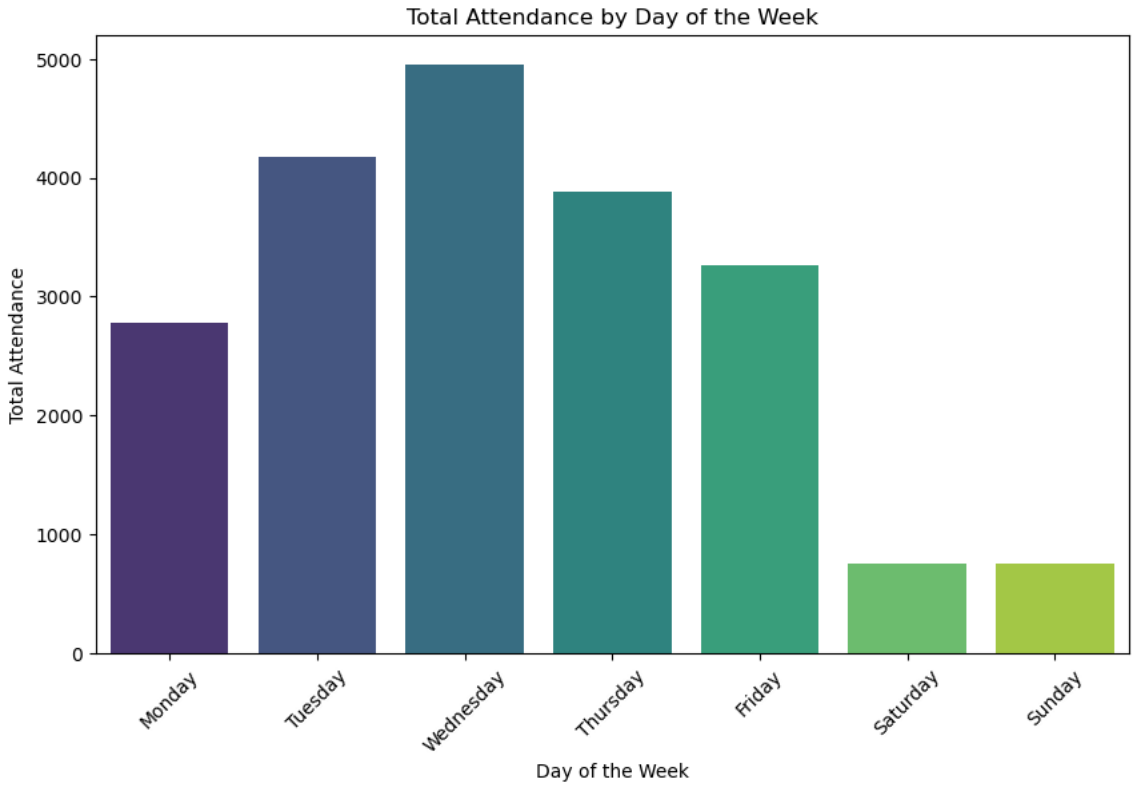
4. Patterns, trends, and insights

Substantial number of top classes face challenges with cancellation, no-shows and having waiting lists as seen in graph below.

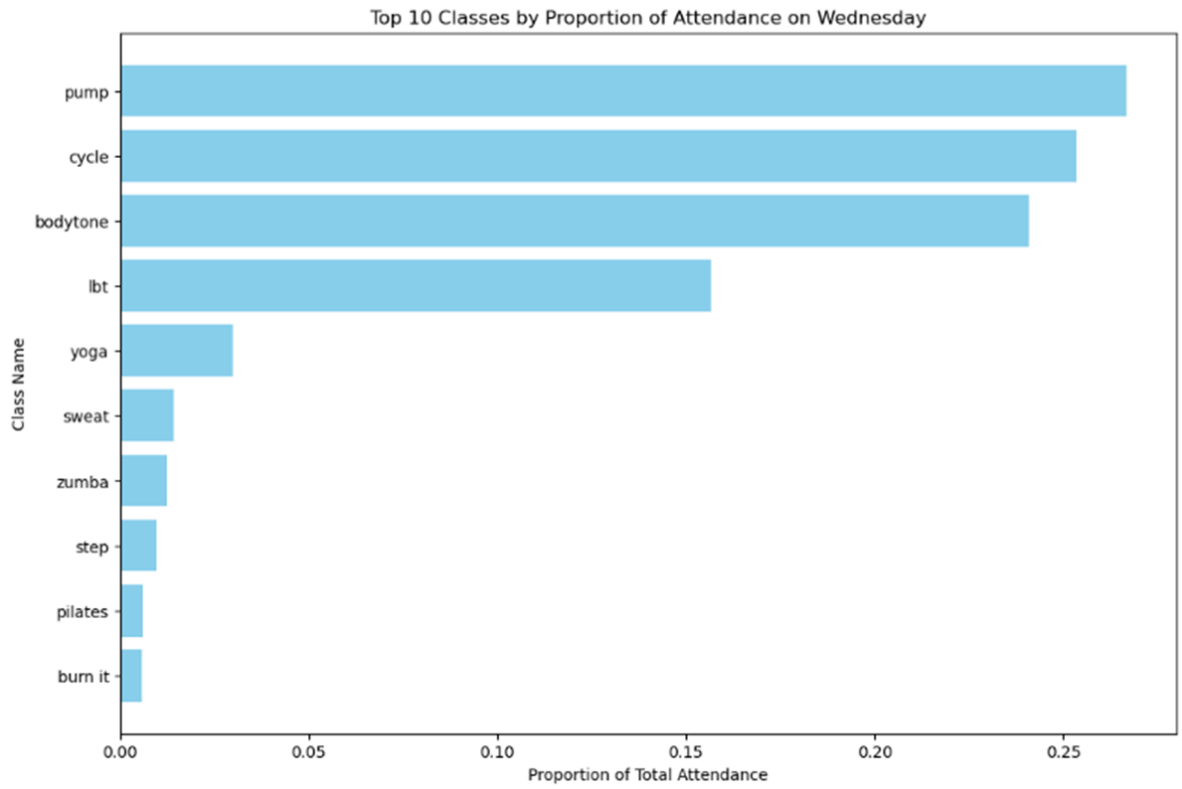


Popular classes like "Cycle" and "Leg Bum & Tums" (LBT) are the most affected. Trend for classes with higher cancellation rates also have moderately high numbers of non-attendance rates. This indicates potential dissatisfaction by consumers due to missed opportunities to maximise class attendance. Significant missed engagements leaves sessions inefficiently used which could have been allocated to eager members on the waitlists. High reservations indicate the need for improved management of bookings and cancellations.

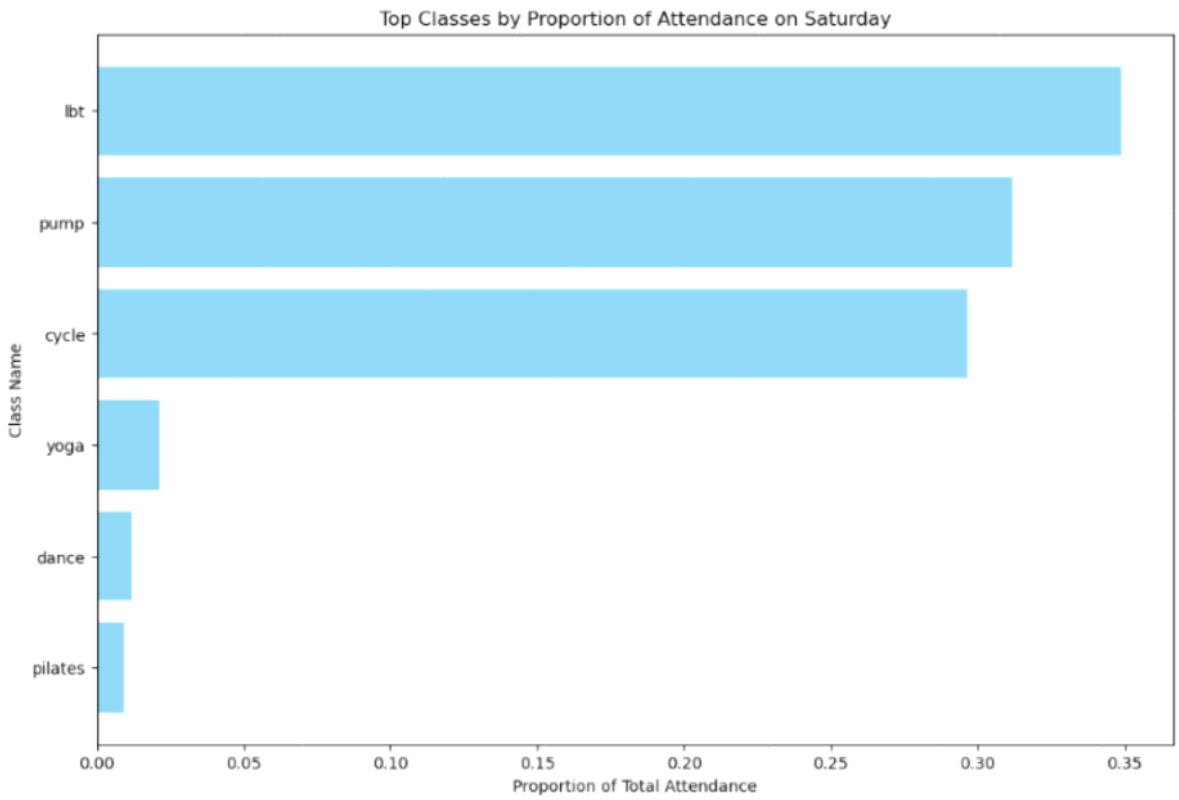
Wednesday has the highest attendance rate, exceeding 20%, while other days fall below 20%. Attendance on Saturdays and Sundays is significantly lower compared to weekdays (Monday to Friday) with under 5% . This suggests that members have fewer commitments on weekends, therefore, introducing more appointments is feasible on Wednesday.

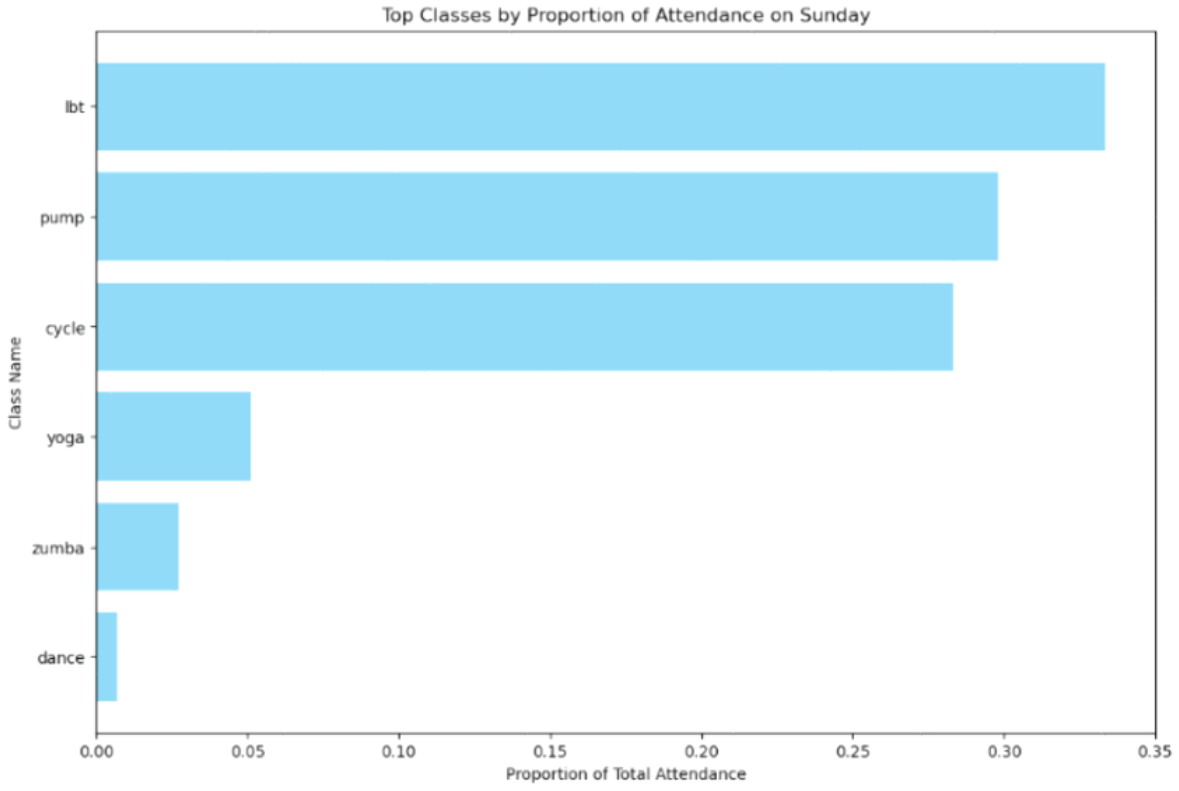


A further examination of the scheduling done on Wednesday shows that Pump, Cycle, and Bodytone are the most popular classes while Yoga, Sweat, Zumba, Step, Pilates, and Burn It have relatively lower attendance proportions.

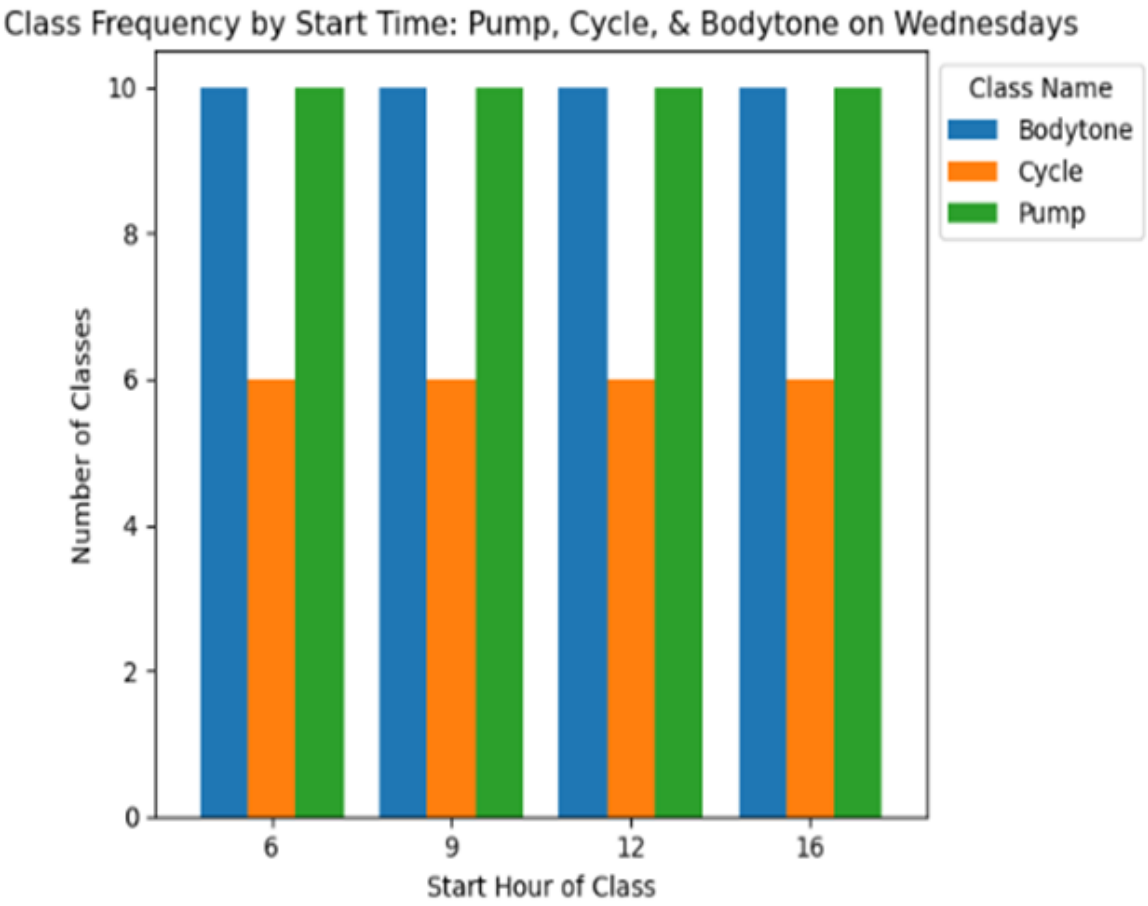


In return, on Saturday and Sunday, the highly popular Bodytone class is not available on weekends. Zumba is not offered on Saturdays, same for Pilates on Sundays. Yoga, Zumba, Dance, and Pilates continue to exhibit low attendance proportions. This offers a chance to identify minimally used lessons that can be swapped or done away with all together.


Top Classes by Proportion of Attendance on Saturday
lbt
pump
cycle
yoga
dance
pilates
Class Name
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
Proportion of Total Attendance

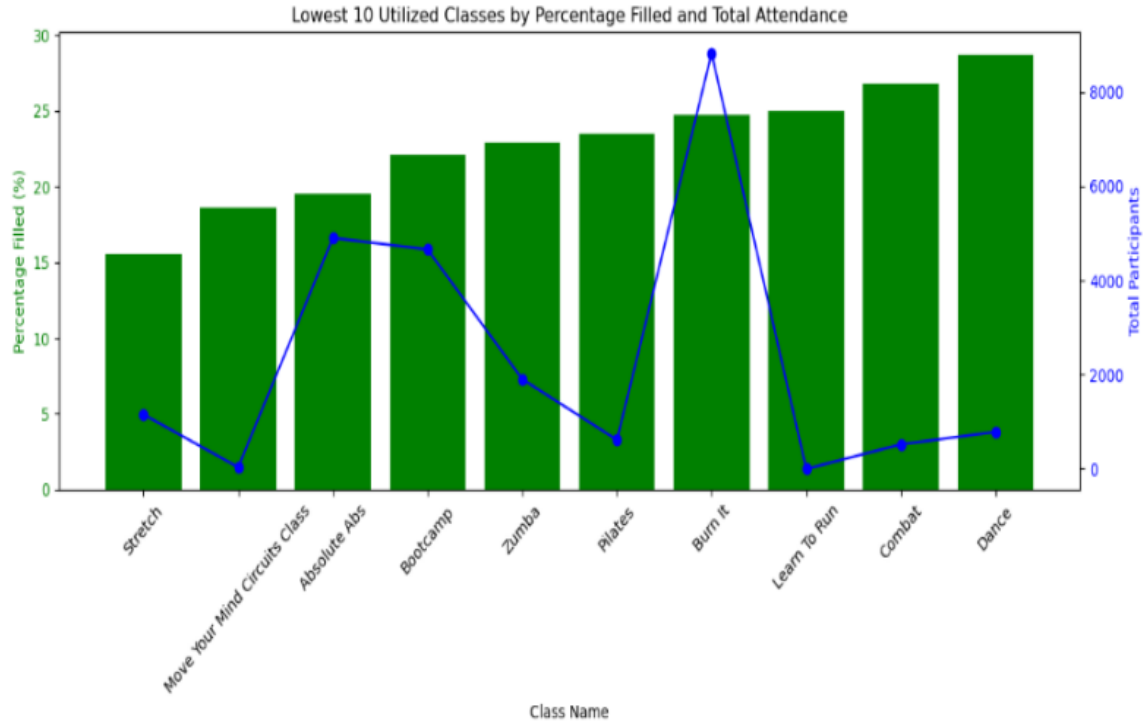

Top Classes by Proportion of Attendance on Sunday
lbt
pump
cycle
yoga
zumba
dance
Class Name
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
Proportion of Total Attendance

Previous analysis suggests that adding extra classes on weekends may not be advisable hence focus is made on how to increase it on Wednesday. The following graph depicts the schedule of the top 3 classes.



Exclude Cycle because it has its own studio, which means it can have unlimited sessions without interference from other programs. LBT is significantly less available at 12 PM, with only 1 session.

Percentage filled and total participants analysis showed;



From this, suggestions for class schedule changes include;

12 PM Slot Adjustment Reduced Step from 10 to 4 sessions, reallocating 6 sessions to LBT, increasing its availability to 7 sessions.

Consider Removing ‘Move Your Mind Circuits Class’ (18.59% filled, 37 participants) and ‘Learn To Run’ (25% filled, 2 participants).

Further exploration is required to track the impact of implemented changes on member satisfaction, retention, and overall gym performance by collecting reviews data. Continuously monitor data to identify new opportunities for further optimization and ensure long-term success by adding a segment online and in PureGym application for clients to provide reasons for cancelling classes.

5. Appendix

A. Import the required libraries in Python.

```
# Import all the necessary packages.
import numpy as np
import pandas as pd
import statsmodels.api as sm
import statsmodels.stats.api as sms
import sklearn
import matplotlib.pyplot as plt
import seaborn as sns
```

B. Code to clean Class data

```
# Load the CSV file
class_data = pd.read_csv('class_data.csv')

# Display the first 5 rows of the DataFrame
class_data.head()
```

```
# Display information about the DataFrame
class_data.info()
```

```
# Show descriptive statistics
class_data.describe()
```

```
# Check for missing values
missing_values = class_data.isnull().sum()
print("Missing values in each column:\n", missing_values)
```

```
# Check column names before dropping
print(class_data.columns.tolist())  # View list of columns
```

```
# Remove redundant columns
cleaned_class_data = class_data.drop(columns=['unknown','start_date_time','stop_date_time'])
```

```
# View list of columns
print(cleaned_class_data.columns.tolist())
```

```
# Display the first few rows of the cleaned DataFrame to verify
print(cleaned_class_data.head())
```

```
# Display information about the cleaned DataFrame
cleaned_class_data.info()
```

```
# Check for duplicates
duplicate_rows = class_data.duplicated()
num_duplicates_class_data = duplicate_rows.sum()
num_duplicates_class_data
```

```
# Remove rows where participation_status is 'BOOKED' or 'WAITING LIST'
cleaned_class_data = cleaned_class_data[~cleaned_class_data['participation_status'].isin(['BOOKED', 'WAITING LIST'])]
```

```
# Standardize class names by stripping whitespace, converting to lowercase, and removing duplicates
cleaned_class_data['class_name'] = cleaned_class_data['class_name'].str.lower().str.strip()
```

```
cleaned_class_data.head()
```

```
# Save the cleaned DataFrame to a new CSV file
class_data.to_csv('cleaned_class_data.csv', index=False)

# Confirm that the CSV file is created
import os
if os.path.exists('cleaned_class_data.csv'):
    print("CSV file successfully created.")
else:
    print("Error: CSV file creation failed.")
```

C. Code to clean Gym data

```
# Load the CSV file
gym_data = pd.read_csv('gym_data.csv')

# Display the first 5 rows of the DataFrame
gym_data.head()
```

```
# Display information about the DataFrame
gym_data.info()

# Show descriptive statistics
gym_data.describe()
```

```
# Check for missing values
missing_values = gym_data.isnull().sum()
print("Missing values in each column:\n", missing_values)
```

```
# Drop the specified columns
cleaned_gym_data = gym_data.drop(columns=['format','offpeak_MF_1','offpeak_MF_2','offpeak_MF_3', 'offpeak_MF_4','offpeak_SS'])
```

```
 # View list of columns
print(cleaned_gym_data.columns.tolist())
```

```
# Display the first few rows of the cleaned DataFrame to verify
print(cleaned_gym_data.head())
```

```
# Display information about the cleaned DataFrame
cleaned_gym_data.info()
```

```
# Check for duplicates
duplicate_rows = cleaned_gym_data.duplicated()
num_duplicates_cleaned_gym_data = duplicate_rows.sum()
num_duplicates_cleaned_gym_data
```

```
cleaned_gym_data.head()
```

```
# Save the cleaned DataFrame to a new CSV file
gym_data.to_csv('cleaned_gym_data.csv', index=False)

# Confirm that the CSV file is created
import os
if os.path.exists('cleaned_gym_data.csv'):
    print("CSV file successfully created.")
else:
    print("Error: CSV file creation failed.")
```

D. Code to clean Visit Data

```
# Load the CSV file
visit_data = pd.read_csv('visit_data.csv')

# Display the first 5 rows of the DataFrame
visit_data.head()
```

```
# Display information about the DataFrame
visit_data.info()

# Show descriptive statistics
visit_data.describe()
```

```
# Check for missing values
missing_values = visit_data.isnull().sum()
print("Missing values in each column:\n", missing_values)
```

```
# Drop the specified columns
cleaned_visit_data = visit_data.drop(columns=['visit_key','check_out_datetime','check_in_result'])
```

```
# Display the first few rows of the cleaned DataFrame to verify
print(cleaned_visit_data.head())
```

```
# Display information about the cleaned DataFrame
cleaned_visit_data.info()
```

```
# Check for duplicates
duplicate_rows = cleaned_visit_data.duplicated()
num_duplicates_cleaned_visit_data = duplicate_rows.sum()
num_duplicates_cleaned_visit_data
```

```
# Remove duplicates
cleaned_visit_data.drop_duplicates(inplace=True)
```

```
# Convert the check_in_datetime to a datetime object
cleaned_visit_data['check_in_datetime'] = pd.to_datetime(cleaned_visit_data['check_in_datetime'])

# Optionally, format the datetime to a more readable string format
# For example, 'YYYY-MM-DD HH:MM:SS'
cleaned_visit_data['check_in_datetime'] = cleaned_visit_data['check_in_datetime'].dt.strftime('%d-%m-%Y %H:%M:%S')

print(cleaned_visit_data['check_in_datetime'])
```

```
# Remove rows where gender is 'unknown'
cleaned_visit_data = visit_data[visit_data['gender'].str.lower() != 'unknown']
```

```
cleaned_visit_data.head()
```

```
# Save the cleaned DataFrame to a new CSV file
visit_data.to_csv('cleaned_visit_data.csv', index=False)

# Confirm that the CSV file is created
import os
if os.path.exists('cleaned_visit_data.csv'):
    print("CSV file successfully created.")
else:
    print("Error: CSV file creation failed.")
```

E. Code to clean Timetable Data

```
# Load the CSV file
timetable_data = pd.read_csv('timetable_data.csv')

# Display the first 5 rows of the DataFrame
timetable_data.head()
```

```
# Display information about the DataFrame
timetable_data.info()

# Show descriptive statistics
timetable_data.describe()
```

```
# Check for missing values
missing_values = timetable_data.isnull().sum()
print("Missing values in each column:\n", missing_values)
```

```
# Check for duplicates
duplicate_rows = timetable_data.duplicated()
num_duplicates_timetable_data = duplicate_rows.sum()
num_duplicates_timetable_data
```

F. Load the cleaned data

```
# Load the cleaned data
cleaned_class_data = pd.read_csv('cleaned_class_data.csv')
cleaned_gym_data = pd.read_csv('cleaned_gym_data.csv')
cleaned_visit_data = pd.read_csv('cleaned_visit_data.csv')
```

G. Class Participation Status Analysis

```
# Standardize class names
cleaned_class_data['class_name'] = cleaned_class_data['class_name'].str.strip().str.title()

# Group by class_name and participation_status, then count the occurrences
class_participation_summary = cleaned_class_data.groupby(['class_name', 'participation_status']).size().unstack(fill_value=0)

# Aggregate participation status by class_name
participation_agg = class_participation_summary

# Find the top 7 classes based on total participation
top_classes = participation_agg.sum(axis=1).sort_values(ascending=False).head(7).index

# Filter data to include only the top 7 classes
participation_agg_top = participation_agg.loc[top_classes]

# Plot stacked bar chart for participation status in top 5 classes
participation_agg_top.plot(kind='bar', stacked=True, figsize=(10, 7), colormap='viridis')
plt.title('Participation Status by Top 7 Classes')
plt.xlabel('Class Name')
plt.ylabel('Number of Participants')
plt.xticks(rotation=45)
plt.legend(title='Participation Status')
plt.show()
```

```python
# Top 7 classes by no-shows
top_7_no_shows = class_participation_summary['NO SHOW'].nlargest(7)
print("\nTop 7 Classes by No-Shows:")
print(top_7_no_shows)

# Top 7 classes by waiting list
top_7_waiting_list = class_participation_summary['WAITING LIST'].nlargest(7)
print("\nTop 7 Classes by Waiting List:")
print(top_7_waiting_list)

# Top 7 classes by cancellations
top_7_cancellations = class_participation_summary['CANCELLED'].nlargest(7)
print("\nTop 7 Classes by Cancellations:")
print(top_7_cancellations)

# Get the union of the top 7 classes by cancellations, no-shows, and waiting list
top_classes = top_7_no_shows.index.union(top_7_waiting_list.index).union(top_7_cancellations.index)

# Get the values for cancellations, no-shows, and waiting list for these classes
cancellations = class_participation_summary.loc[top_classes, 'CANCELLED']
no_shows = class_participation_summary.loc[top_classes, 'NO SHOW']
waiting_list = class_participation_summary.loc[top_classes, 'WAITING LIST']

# Calculate total occurrences for normalization
total_occurrences = cancellations + no_shows + waiting_list

# Calculate proportions
cancellations_proportion = (cancellations / total_occurrences) * 100
no_shows_proportion = (no_shows / total_occurrences) * 100
waiting_list_proportion = (waiting_list / total_occurrences) * 100

# Plotting
x = np.arange(len(top_classes))  # the label locations
width = 0.25  # the width of the bars

plt.figure(figsize=(14, 7))D
plt.bar(x - width, no_shows_proportion, width, label='No-Shows', color='blue')
plt.bar(x, cancellations_proportion, width, label='Cancellations', color='green')
plt.bar(x + width, waiting_list_proportion, width, label='Waiting List', color='orange')

plt.title('Proportion of No-Shows, Cancellations, and Waiting List by Class')
plt.xlabel('Class Name')
plt.ylabel('Proportion of Occurrences (%)')
plt.xticks(x, top_classes, rotation=45)
plt.legend()

plt.tight_layout()
plt.show()
```

H. Day of Week with Attendance Analysis

```
# Filter for 'ATTENDED' status in cleaned_class_data
attended_data = cleaned_class_data[cleaned_class_data['participation_status'] == 'ATTENDED']

# Calculate total participants for each class
attended_data['total_participants'] = attended_data['female'] + attended_data['male']

# Merge the timetable data with the attendance data on 'class_name'
merged_data = pd.merge(timetable_data, attended_data, on='class_name', how='left')

# Calculate total attendance per day of the week
attendance_by_day = merged_data.groupby('day_of_week')['total_participants'].sum().reset_index(name='total_attendance')

# Calculate total attendance across all days
total_attendance = attendance_by_day['total_attendance'].sum()

# Calculate percentage attendance per day
attendance_by_day['percentage_attendance'] = (attendance_by_day['total_attendance'] / total_attendance) * 100

# Ensure days of the week are in the correct order
day_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
attendance_by_day['day_of_week'] = pd.Categorical(attendance_by_day['day_of_week'], categories=day_order, ordered=True)
attendance_by_day = attendance_by_day.sort_values('day_of_week')

# Display the attendance by day
print(attendance_by_day[['day_of_week', 'percentage_attendance']])

# Plotting
plt.figure(figsize=(10, 6))
sns.barplot(x='day_of_week', y='percentage_attendance', data=attendance_by_day, palette='viridis')
plt.title('Percentage Attendance by Day of the Week')
plt.xlabel('Day of the Week')
plt.ylabel('Percentage Attendance (%)')
plt.xticks(rotation=45)
plt.show()
```

```
cleaned_class_data['class_name'] = cleaned_class_data['class_name'].str.strip().str.title()
timetable_data['class_name'] = timetable_data['class_name'].str.strip().str.title()

print(merged_data.isnull().sum())

merged_data = pd.merge(cleaned_class_data, timetable_data, on='class_name', how='outer')
```

```
# Assuming 'class_name' is the common key
merged_data = pd.merge(cleaned_class_data, timetable_data, on='class_name', how='left')

# Filter for 'ATTENDED' status
attended_data = merged_data[merged_data['participation_status'] == 'ATTENDED']

# Calculate total participants for each class
attended_data['total_participants'] = attended_data['female'] + attended_data['male']

# Filter the dataset for Wednesday
wednesday_data = attended_data[attended_data['day_of_week'] == 'Wednesday']

# Aggregate the total attendance by class for Wednesday
wednesday_attendance = wednesday_data.groupby('class_name')['total_participants'].sum().reset_index()

# Calculate total attendance across all classes on Wednesday
total_attendance_wednesday = wednesday_attendance['total_participants'].sum()

# Calculate the proportion of attendance for each class
wednesday_attendance['attendance_proportion'] = wednesday_attendance['total_participants'] / total_attendance_wednesday

# Sort the data to find the top 10 classes with the highest attendance proportion
top_7_wednesday_classes = wednesday_attendance.sort_values(by='attendance_proportion', ascending=False).head(13)

# Print top 5 data
print(top_7_wednesday_classes)

# Plot the top 5 classes as proportions
plt.figure(figsize=(12, 8))
plt.barh(top_7_wednesday_classes['class_name'], top_7_wednesday_classes['attendance_proportion'], color='skyblue')
plt.xlabel('Proportion of Total Attendance')
plt.ylabel('Class Name')
plt.title('Classes by Proportion of Attendance on Wednesday')
plt.gca().invert_yaxis()  # Invert y-axis to show the highest at the top
plt.show()
```

```python
# Assuming 'class_name' is the common key
merged_data = pd.merge(cleaned_class_data, timetable_data, on='class_name', how='left')

# Filter for 'ATTENDED' status
attended_data = merged_data[merged_data['participation_status'] == 'ATTENDED']

# Calculate total participants for each class
attended_data['total_participants'] = attended_data['female'] + attended_data['male']

# Filter the dataset for Saturday
saturday_data = attended_data[attended_data['day_of_week'] == 'Saturday']

# Aggregate the total attendance by class for Saturday
saturday_attendance = saturday_data.groupby('class_name')['total_participants'].sum().reset_index()

# Calculate total attendance across all classes on Saturday
total_attendance_saturday = saturday_attendance['total_participants'].sum()

# Calculate the proportion of attendance for each class
saturday_attendance['attendance_proportion'] = saturday_attendance['total_participants'] / total_attendance_saturday

# Sort the data to find the top 5 classes with the highest attendance proportion
top_saturday_classes = saturday_attendance.sort_values(by='attendance_proportion', ascending=False).head(7)

# Print top  data
print(top_saturday_classes)

# Plot the top classes as proportions
plt.figure(figsize=(12, 8))
plt.barh(top_saturday_classes['class_name'], top_saturday_classes['attendance_proportion'], color='skyblue')
plt.xlabel('Proportion of Total Attendance')
plt.ylabel('Class Name')
plt.title('Top Classes by Proportion of Attendance on Saturday')
plt.gca().invert_yaxis()  # Invert y-axis to show the highest at the top
plt.show()
```

I. Class Frequency and Suggestion

```
# Assuming 'class_name' is the common key
merged_data = pd.merge(cleaned_class_data, timetable_data, on='class_name', how='left')

# Filter for 'ATTENDED' status
attended_data = merged_data[merged_data['participation_status'] == 'ATTENDED']

# Calculate total participants for each class
attended_data['total_participants'] = attended_data['female'] + attended_data['male']

# Filter the dataset for Sunday
sunday_data = attended_data[attended_data['day_of_week'] == 'Sunday']

# Aggregate the total attendance by class for Sunday
sunday_attendance = sunday_data.groupby('class_name')['total_participants'].sum().reset_index()

# Calculate total attendance across all classes on Sunday
total_attendance_sunday = sunday_attendance['total_participants'].sum()

# Calculate the proportion of attendance for each class
sunday_attendance['attendance_proportion'] = sunday_attendance['total_participants'] / total_attendance_sunday

# Sort the data to find the top 10 classes with the highest attendance proportion
top_sunday_classes = sunday_attendance.sort_values(by='attendance_proportion', ascending=False).head(10)

# Print top data
print(top_sunday_classes)

# Plot the top classes as proportions
plt.figure(figsize=(12, 8))
plt.barh(top_sunday_classes['class_name'], top_sunday_classes['attendance_proportion'], color='skyblue')
plt.xlabel('Proportion of Total Attendance')
plt.ylabel('Class Name')
plt.title('Top Classes by Proportion of Attendance on Sunday')
plt.gca().invert_yaxis()  # Invert y-axis to show the highest at the top
plt.show()
# Filter for Wednesday classes and the specific class names in timetable_data
wednesday_classes = timetable_data[
    (timetable_data['day_of_week'] == 'Wednesday') &
    (timetable_data['class_name'].isin(['Pump', 'Lbt', 'Bodytone']))
]

# Ensure 'slot_start_time' is a datetime type to extract the hour
wednesday_classes['slot_start_time'] = pd.to_datetime(wednesday_classes['slot_start_time'])
wednesday_classes['start_hour'] = wednesday_classes['slot_start_time'].dt.hour

# Group by class name and start hour and count the occurrences
wednesday_class_distribution = wednesday_classes.groupby(['class_name', 'start_hour']).size().reset_index(name='class_count')

# Pivot the table to get class names as columns and start hours as rows
pivot_table = wednesday_class_distribution.pivot(index='start_hour', columns='class_name', values='class_count').fillna(0)

# Plotting the distribution of classes by start hour for each class
plt.figure(figsize=(14, 7))
pivot_table.plot(kind='bar', stacked=False, width=0.8)
plt.title('Wednesday Schedule Distribution for the Top 3 Popular Classes')
plt.xlabel('Start Hour of Class')
plt.ylabel('Number of Classes')
plt.xticks(rotation=0)

# Position legend outside of the plot
plt.legend(title='Class Name', bbox_to_anchor=(1.05, 1), loc='upper left')

# Adjust layout
plt.tight_layout()

# Display the plot
plt.show()

print(pivot_table)
```

```python
# Filter for Wednesday classes in timetable_data
wednesday_classes = timetable_data[timetable_data['day_of_week'] == 'Wednesday']

# Ensure 'slot_start_time' is a datetime type to extract the hour
wednesday_classes['slot_start_time'] = pd.to_datetime(wednesday_classes['slot_start_time'])
wednesday_classes['start_hour'] = wednesday_classes['slot_start_time'].dt.hour

# Filter for classes that start at 6 AM and 12 PM
wednesday_classes_filtered = wednesday_classes[wednesday_classes['start_hour'].isin([6, 12])]

# Group by class name and start hour and count the occurrences
wednesday_class_distribution = wednesday_classes_filtered.groupby(['class_name', 'start_hour']).size().reset_index(name='class_c

# Pivot the table to get class names as columns and start hours as rows
pivot_table = wednesday_class_distribution.pivot(index='start_hour', columns='class_name', values='class_count').fillna(0)

# Remove the 'Cycle' column
pivot_table = pivot_table.drop(columns=['Cycle'], errors='ignore')

# Plotting the distribution of classes starting at 6 AM
if 6 in pivot_table.index:
    plt.figure(figsize=(10, 6))
    pivot_table.loc[6].plot(kind='bar', stacked=False, width=0.8)
    plt.title('Class Frequency by Start Time (6 AM) for All Classes on Wednesdays (Excluding Cycle)')
    plt.xlabel('Class Name')
    plt.ylabel('Number of Classes')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()

# Plotting the distribution of classes starting at 12 PM
if 12 in pivot_table.index:
    plt.figure(figsize=(10, 6))
    pivot_table.loc[12].plot(kind='bar', stacked=False, width=0.8)
    plt.title('Class Frequency by Start Time (12 PM) for All Classes on Wednesdays (Excluding Cycle)')
    plt.xlabel('Class Name')
    plt.ylabel('Number of Classes')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()

print(pivot_table)
```

```python
# Assuming the pivot_table data is as provided:
data = {
    'class_name': ['Bodytone', 'Burn It', 'Lbt', 'Pilates', 'Pump', 'Step', 'Sweat', 'Yoga'],
    '6': [10.0, 2.0, 0.0, 10.0, 10.0, 0.0, 0.0, 6.0],
    '12': [10.0, 0.0, 1.0, 0.0, 10.0, 10.0, 7.0, 10.0]
}
pivot_table = pd.DataFrame(data).set_index('class_name').T

# Adjust the data based on the user's request
pivot_table.at['6', 'Pilates'] = 3
pivot_table.at['6', 'Lbt'] = 7
pivot_table.at['12', 'Step'] = 4
pivot_table.at['12', 'Lbt'] = 7

# Plotting the distribution of classes starting at 6 AM
if '6' in pivot_table.index:
    plt.figure(figsize=(10, 6))
    pivot_table.loc['6'].plot(kind='bar', stacked=False, width=0.8)
    plt.title('Suggested Class Frequency for 6 AM on Wednesdays')
    plt.xlabel('Class Name')
    plt.ylabel('Number of Classes')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()

# Plotting the distribution of classes starting at 12 PM
if '12' in pivot_table.index:
    plt.figure(figsize=(10, 6))
    pivot_table.loc['12'].plot(kind='bar', stacked=False, width=0.8)
    plt.title('Suggested Class Frequency for 12 PM on Wednesdays')
    plt.xlabel('Class Name')
    plt.ylabel('Number of Classes')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()

# Display the adjusted data
print(pivot_table)
```

J. Class Capacity Utilisation

```
# Filter for Sunday classes with 'ATTENDED' status
sunday_classes = merged_data[
    (merged_data['day_of_week'] == 'Sunday') &
    (merged_data['participation_status'] == 'ATTENDED')
]

# Calculate total attendees (sum of male and female)
sunday_classes['total_attended'] = sunday_classes['male'] + sunday_classes['female']

# Aggregate data by class name and calculate averages
class_capacity_averages_sunday = sunday_classes.groupby('class_name').agg(
    average_attended=pd.NamedAgg(column='total_attended', aggfunc='mean'),
    average_capacity=pd.NamedAgg(column='class_capacity', aggfunc='mean')
).reset_index()

# Calculate the utilization percentage
class_capacity_averages_sunday['utilization_rate'] = (class_capacity_averages_sunday['average_attended'] / class_capacity_averag

# Calculate the remaining capacity
class_capacity_averages_sunday['remaining_capacity'] = class_capacity_averages_sunday['average_capacity'] - class_capacity_avera

# Plotting
fig, ax = plt.subplots(figsize=(16, 10))

# Stacked Bar Chart
bars = ax.bar(class_capacity_averages_sunday['class_name'],
              class_capacity_averages_sunday['average_attended'],
              color='green', label='Average Attended')

remaining_capacity = ax.bar(class_capacity_averages_sunday['class_name'],
                            class_capacity_averages_sunday['remaining_capacity'],
                            bottom=class_capacity_averages_sunday['average_attended'],
                            color='grey', label='Remaining Capacity')

# Add labels and title
ax.set_xlabel('Class Name', fontsize=14)
ax.set_ylabel('Number of Participants', fontsize=14)
ax.set_title('Average Class Capacity Utilisation on Sundays', fontsize=16)
plt.xticks(rotation=45, fontsize=12)
plt.yticks(fontsize=12)

# Annotate bars with utilization rate
for i in range(len(class_capacity_averages_sunday)):
    ax.text(i, class_capacity_averages_sunday['average_attended'][i] + 1,
            f'{class_capacity_averages_sunday["utilization_rate"][i]:.1f}%',
            ha='center', va='bottom', color='black', fontsize=12)

# Show legend
ax.legend(loc='upper left', bbox_to_anchor=(1, 1), fontsize=12)

# Adjust layout to avoid clipping
plt.tight_layout()

# Display the plot
plt.show()
```

```python
# Filter for Saturday classes with 'ATTENDED' status
saturday_classes = merged_data[
    (merged_data['day_of_week'] == 'Saturday') &
    (merged_data['participation_status'] == 'ATTENDED')
]

# Calculate total attendees (sum of male and female)
saturday_classes['total_attended'] = saturday_classes['male'] + saturday_classes['female']

# Aggregate data by class name and calculate averages
class_capacity_averages_saturday = saturday_classes.groupby('class_name').agg(
    average_attended=pd.NamedAgg(column='total_attended', aggfunc='mean'),
    average_capacity=pd.NamedAgg(column='class_capacity', aggfunc='mean')
).reset_index()

# Calculate the utilization percentage
class_capacity_averages_saturday['utilization_rate'] = (class_capacity_averages_saturday['average_attended'] / class_capacity_av

# Calculate the remaining capacity
class_capacity_averages_saturday['remaining_capacity'] = class_capacity_averages_saturday['average_capacity'] - class_capacity_a

# Plotting
fig, ax = plt.subplots(figsize=(16, 10))

# Stacked Bar Chart
bars = ax.bar(class_capacity_averages_saturday['class_name'],
              class_capacity_averages_saturday['average_attended'],
              color='green', label='Average Attended')

remaining_capacity = ax.bar(class_capacity_averages_saturday['class_name'],
                            class_capacity_averages_saturday['remaining_capacity'],
                            bottom=class_capacity_averages_saturday['average_attended'],
                            color='grey', label='Remaining Capacity')

# Add labels and title
ax.set_xlabel('Class Name', fontsize=14)
ax.set_ylabel('Number of Participants', fontsize=14)
ax.set_title('Average Class Capacity Utilization on Saturdays', fontsize=16)
plt.xticks(rotation=45, fontsize=12)
plt.yticks(fontsize=12)

# Annotate bars with utilization rate
for i in range(len(class_capacity_averages_saturday)):
    ax.text(i, class_capacity_averages_saturday['average_attended'][i] + 1,
            f'{class_capacity_averages_saturday["utilization_rate"][i]:.1f}%',
            ha='center', va='bottom', color='black', fontsize=12)

# Show legend
ax.legend(loc='upper left', bbox_to_anchor=(1, 1), fontsize=12)

# Adjust layout to avoid clipping
plt.tight_layout()

# Display the plot
plt.show()

# Print the utilization rates
print("Class Utilization Rates on Saturday:")
print(class_capacity_averages_saturday[['class_name', 'utilization_rate']])
```

K. Distribution of Check in Times on Days

```
# Convert 'check_in_datetime' to datetime format
cleaned_visit_data['check_in_datetime'] = pd.to_datetime(cleaned_visit_data['check_in_datetime'], errors='coerce')

# Filter for Saturdays
saturday_data = cleaned_visit_data[cleaned_visit_data['check_in_datetime'].dt.dayofweek == 5]

# Filter for Sundays
sunday_data = cleaned_visit_data[cleaned_visit_data['check_in_datetime'].dt.dayofweek == 6]

# Filter for Wednesdays
wednesday_data = cleaned_visit_data[cleaned_visit_data['check_in_datetime'].dt.dayofweek == 2]

# Extract the hour of check-in for each day
saturday_data['check_in_hour'] = saturday_data['check_in_datetime'].dt.hour + saturday_data['check_in_datetime'].dt.minute / 60.
sunday_data['check_in_hour'] = sunday_data['check_in_datetime'].dt.hour + sunday_data['check_in_datetime'].dt.minute / 60.0
wednesday_data['check_in_hour'] = wednesday_data['check_in_datetime'].dt.hour + wednesday_data['check_in_datetime'].dt.minute /

# Plot the distribution of check-in times on Saturdays
plt.figure(figsize=(10, 6))
plt.hist(saturday_data['check_in_hour'], bins=24, range=(0, 24), color='lightgreen', edgecolor='black')
plt.title('Distribution of Check-In Times on Saturdays')
plt.xlabel('Time of Day (Hours)')
plt.ylabel('Number of Check-Ins')
plt.xticks(range(0, 25))  # Set x-ticks to represent hours
plt.grid(True)
plt.show()

# Plot the distribution of check-in times on Sundays
plt.figure(figsize=(10, 6))
plt.hist(sunday_data['check_in_hour'], bins=24, range=(0, 24), color='lightblue', edgecolor='black')
plt.title('Distribution of Check-In Times on Sundays')
plt.xlabel('Time of Day (Hours)')
plt.ylabel('Number of Check-Ins')
plt.xticks(range(0, 25))  # Set x-ticks to represent hours
plt.grid(True)
plt.show()

# Plot the distribution of check-in times on Wednesdays
plt.figure(figsize=(10, 6))
plt.hist(wednesday_data['check_in_hour'], bins=24, range=(0, 24), color='lightcoral', edgecolor='black')
plt.title('Distribution of Check-In Times on Wednesdays')
plt.xlabel('Time of Day (Hours)')
plt.ylabel('Number of Check-Ins')
plt.xticks(range(0, 25))  # Set x-ticks to represent hours
plt.grid(True)
plt.show()
```

L. Lowest 10 Utilised Classes by Percentage Filled and Total Attendance

```
# Filter data for 'ATTENDED' status
attended_data = cleaned_class_data[cleaned_class_data['participation_status'] == 'ATTENDED']

# Calculate total participants for each class
attended_data['total_participants'] = attended_data['female'] + attended_data['male']

# Aggregate total attendance and class capacity for each class
aggregated_data = attended_data.groupby('class_name').agg({
    'total_participants': 'sum',
    'class_capacity': 'sum'  # Summing capacity in case of multiple entries
}).reset_index()

# Calculate percentage of class capacity filled
aggregated_data['percentage_filled'] = (aggregated_data['total_participants'] / aggregated_data['class_capacity']) * 100

# Sort by percentage filled in ascending order to find lowest utilization
class_data_sorted = aggregated_data.sort_values(by='percentage_filled').head(10)

# Plotting based on the order of percentage filled
fig, ax1 = plt.subplots(figsize=(12, 6))

# Plotting the percentage filled on the left y-axis
ax1.bar(class_data_sorted['class_name'], class_data_sorted['percentage_filled'], color='green', label='Percentage Filled')
ax1.set_xlabel('Class Name')
ax1.set_ylabel('Percentage Filled (%)', color='green')
ax1.tick_params(axis='y', labelcolor='green')
ax1.set_xticklabels(class_data_sorted['class_name'], rotation=45)

# Create a second y-axis for total participants
ax2 = ax1.twinx()
ax2.plot(class_data_sorted['class_name'], class_data_sorted['total_participants'], color='blue', marker='o', linestyle='-', labe
ax2.set_ylabel('Total Participants', color='blue')
ax2.tick_params(axis='y', labelcolor='blue')

# Title and layout
plt.title('Lowest 10 Utilised Classes by Percentage Filled and Total Attendance')
fig.tight_layout()

plt.show()

# Display the sorted data for reference
print("Lowest 10 Utilized Classes:")
print(class_data_sorted)
```